REPORT DOCUMENTATION PAGE

OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
		Final

4. TITLE AND SUBTITLE

Active Heterogeneous Databases for Monitor & Control

5. FUNDING NUMBERS

F 49620-93-1-0059

6. AUTHOR(S)

Ouri Wolfson, A. Prasad Sistla, Clement Yu

AFOSR-TR-96

0329

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

University of Illinois at Chicago
Department of Electrical Engineering and Computer Science
851 South Morgan Street
Chicago, IL 60680

8. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Air Force Office of Scientific Research
110 Duncan Avenue, Suite B115
Bolling Air Force Base
Washington, DC 20332-0001

nm

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

19960624 239

12a. DISTRIBUTION/AVAILABILITY STATEMENT

UNCL

~~As part of this project, a rule based declarative trigger language employing *temporal logic*~~ has been developed. This language facilitates monitoring of the evolution of a database over time, and permits the specification of appropriate actions that need to be taken whenever the prespecified conditions are satisfied. Each rule has a condition and an action part. Two versions of the rule language have been developed. In the first version, the condition part of the language is specified in *Past Temporal Logic*, and in the second version the condition part is specified in *Future Temporal Logic*. The project has implemented systems that monitor conditions specified in the two temporal logics. In the area of heterogeneous database systems, systems for schema and query translations between object-oriented databases and relational databases have been developed and implemented. These two types of databases are chosen, because relational databases are widely used and object-oriented databases are gaining popularity. With both the schema and query translators, a relational user can access an object-oriented database and an object-oriented user can access a relational database. Parts of the systems have been ported to the Rush Medical Center, and to Hughes Aircraft Co.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Active Heterogenous Databases, Temporal Logic, Rules, Relational Databases, object oriented Databases	12
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT

Active Heterogeneous Databases for Monitor and Control

Ouri Wolfson
A. Prasad Sistla
Clement Yu
Department of Electrical Engineering and Computer Science
University of Illinois at Chicago,
Chicago, Illinois 60607

May 14, 1996

Final Technical Report on the work done under
AFOSR grant no. F49620-93-1-0059

ABSTRACT:

As part of this project, a rule based declarative trigger language employing *temporal logic* has been developed. This language facilitates monitoring of the evolution of a database over time, and permits the specification of appropriate actions that need to be taken whenever the pre-specified conditions are satisfied. Each rule has a condition and an action part. Two versions of the rule language have been developed. In the first version, the condition part of the language is specified in *Past Temporal Logic*, and in the second version the condition part is specified in *Future Temporal Logic*. The project has implemented systems that monitor conditions specified in the two temporal logics. In the area of heterogeneous database systems, systems for schema and query translations between object-oriented databases and relational databases have been developed and implemented. These two types of databases are chosen, because relational databases are widely used and object-oriented databases are gaining popularity. With both the schema and query translators, a relational user can access an object-oriented database and an object-oriented user can access a relational database. Parts of the systems have been ported to the Rush Medical Center, and to Hughes Aircraft Co.

1 Introduction

Modern systems, such as traffic control, securities trading and communication networks, are increasingly dependent on real-time software applications for monitor and control. At the center of such applications usually lies an active database, that represents the status of the system. This database is continuously updated by (often remote) sensors, and the software is expected to respond to predefined conditions. Often these conditions refer to the evolution of the database state over time (i.e. the database history). For example, in securities trading, the system may be requested to alert a trader when the value of a particular stock (given by some database attribute) increases by more than 10% in 15 minutes. We call such conditions temporal triggers, i.e. triggers on the evolution of the database state over time. Existing database management systems do not provide the capability for specifying temporal triggers. In most of them, the condition part of a rule (a rule is a condition-action pair) refers to either the current database state, or to the transition from one database state to the next, but not to a sequence of database states. Furthermore, different parts of the database may be managed by different types of database management systems. For example, one part of the database may be managed by a relational system, while another part of the database may be managed by an object-oriented database system. The main objective of this project is to develop a system for supporting the specification and processing of temporal triggers over a system of heterogeneous databases. Therefore, this project is divided into two parts. The first part deals with the specification and processing of temporal triggers. The second part deals with heterogeneous database systems.

This report is organized as follows. Section 2 describes the first part of the project, i.e. the work on temporal triggers. Section 3 describes the second part of the project, i.e. the work on heterogeneous database management systems. Section 4 briefly describes the software that has been produced and lists the students who have received their graduate degrees by working on the project. Section 5 lists the papers that report on work partially or entirely supported by this grant.

2 Temporal Triggers

We have developed two languages for the specification of temporal triggers. They are based on temporal logic and are called *Future Temporal Logic* (FTL) and *Past Temporal Logic* (PTL). Future Temporal Logic uses temporal operators that refer to the future database states while Past Temporal Logic uses temporal operators that refer to past database states. It is easier to specify certain triggers in FTL, whereas it is easier to specify certain others in PTL. *Temporal Logic* is a formalism for specifying and reasoning about time-varying properties of systems, and therefore it is an appropriate language for specifying temporal triggers. The main feature of FTL and PTL is the use of special operators that apply exclusively to the time dimension. The temporal operators used in FTL are Until , Nexttime , and Eventually , while the ones used in PTL are Since , Lasttime and Previously . Some other examples of trigger conditions that can be specified in our languages are the following: "A tuple that satisfies a certain predicate is added to the database at least 10 minutes before another tuple, that satisfies another condition, is added to the database". Or, "the value of an attribute stays above a certain threshold for at least 10 minutes". Or, "for a period of 10 minutes, two different values in the database increase simultaneously".

Both PTL and FTL allow us to specify conditions that involve both events (e.g., starting of a transaction, login of a user, commit of a transaction, running of a program) and database states,

and their evolution over time.

PTL also allows temporal aggregates, i.e. aggregate functions over time. Therefore, we can specify a condition such as: "The average speed of the aircraft in the last 20 minutes exceeds 1000". Or, "the Dow Jones Industrial Average fell more than 250 points in the last 2 hours." Or, "the average number of widgets in inventory in the last 20 days is below 200".

In addition to the languages, we also proposed efficient algorithms for incremental evaluation of the temporal conditions specified in PTL and FTL respectively. The evaluations are incremental in the sense that when a new database state is created as a result of an update, the algorithms only consider the changes in the new database state in order to determine if the condition is satisfied, instead of considering the whole database history. The evaluation algorithms are also add-on components, executed on top of, and using the existing database management system as follows. Based on a temporal trigger specification, we identify a set of materialized views that should be presented to the DBMS as traditional triggers. Namely, the DBMS is requested to invoke the temporal component whenever any of the materialized views changes. When invoked, the temporal component determines whether or not the trigger is satisfied. If so, then the user is notified, or the associated action is taken. Otherwise, the temporal component saves a minimum amount of information from the current database state, in order to be able to determine in its future invocations whether or not the trigger condition is satisfied.

We have implemented the temporal component for a subset of FTL and for a subset of PTL on top of the Sybase DBMS running on SUN work-stations.

Temporal integrity constraints are temporal conditions that must be satisfied at every commit point of a transaction. We proposed a new framework for processing temporal integrity constraints. Existing methods for processing such constraints translate them into a set of rules with nontemporal conditions. In contrast, we process integrity constraints "from first principles" so to speak. Our rule conditions are temporal, thus we use the algorithm that processes temporal conditions for both, rule processing and integrity constraints processing.

We show that our proposed languages and processing algorithms can work with "valid time" as well as "transaction time". The important distinction between the two notions of time is critical in rule and constraint processing, in the following sense. The database is usually a model of the real-world and the time at which changes happen in the real-world, namely the valid time, may precede the time at which these changes are posted in the database, namely the transaction time. Furthermore, the temporal conditions may refer to the valid time rather than the transaction time. For example, consider a transaction that updates the price of a stock, i.e., posts to the database the price of a stock sale. The sale may have occurred five minutes before the change in the stock price is committed to the database. It is easy to see that there are triggers that will fire with respect to the valid time but will not fire with respect to the transaction time, and vice versa. For example, consider the temporal condition: "the stock price remains constant for seven minutes". This condition may be satisfied with respect to the transaction time, but not with respect to the valid time, and vice versa.

We also showed that in the valid time model temporal conditions in rules and integrity constraints can have different semantics. For example, we showed that an integrity constraint with respect to a database history can be "online" satisfied but not "offline" satisfied, and vice versa. Similarly, we showed that execution of an action in a rule can be based on either "tentative" or "definite" history, and our language and processing algorithm can accommodate both notions.

Our rule formalism also enables the specification of temporal actions, i.e., composite actions built from atomic ones using temporal operators. For example, the user can specify that when a certain condition is satisfied, the system should execute the BUY-STOCK transaction every

10 minutes (in order to prevent driving up the stock-price), as long as the stock-price remains below 50. Previously, such temporal actions required extended transactions models that must be controlled by an additional software system running on top of the database management system. The temporal condition formalism also requires an additional software system, the condition evaluator, but this is the same system used for rule processing, integrity constraint enforcement, and temporal actions. Furthermore, the specification of temporal actions involving iteration and absolute and relative timing properties (every 10 minutes) is not straightforward in existing models of extended transactions.

Due to space limitations, in the remainder of this section, we only give an informal as well as a formal presentation of our language FTL, and illustrate its application through examples. For the description of the language PTL and the algorithms for processing triggers specified in both these languages, the reader is refered to the papers cited at the end of this report.

2.1 The Model

We assume that the database is composed of variables of various types, such as relations, objects, etc. We refer to these variables as *database variables.* A *database state* is a mapping that associates a value from the appropriate domain to each variable. The formulas of the logic FTL are interpreted over database histories. A *database history* is a finite sequence of database states, each with an associated *time stamp.* A new database state is added to the history whenever the database changes. The time stamp associated with the new database state denotes the time at which the change occurs, according to a fixed global clock[1]. We assume that the time stamps along the history are strictly increasing.

This model only refers to the database variables, and it does not explicitly consider external events such as—transaction-begin, transaction-commit, etc. We can easily incorporate the external events into our model by assuming that the database has a special relation which, at any instance of time, contains all the external events that occur at that time.

2.2 Syntax of FTL

The formulas of FTL use two basic future temporal operators Until and Nexttime . Other temporal operators, such as Eventually , can be expressed in terms of the basic operators. The symbols of the logic include various *type* names, such as relations, integers, etc. These denote the different types of the variables and constants in the database. We assume that, for each $n \geq 0$, we have a set of n-ary *function* symbols and a set of n-ary *relation* symbols. Each n-ary function symbol denotes a function that takes n-arguments of particular types, and returns a value. For example, + and * are function symbols denoting addition and multiplication on the integer type. Similarly, $\leq, \geq$ are binary relation symbols denoting arithmetic comparison operators. The functions symbols are also used to denote retrieval queries on databases. For example, consider the following relational query, BUSY-LINKS, that retrieves all busy links from a relation called TRAFFIC-ON-LINK (the relation contains tuples consisting of a link name and the traffic on that link):

RETRIEVE (TRAFFIC-ON-LINK.name) WHERE TRAFFIC-ON-LINK.traffic≥100

[1] Our notion of time is discrete, but with slight modifications we can extend the model to the case of continuous time.

The above query is represented by the function symbol BUSY-LINKS, that takes a single argument which is a binary relation, and returns a unary relation. Functions of arity zero denote constants and relations of arity zero denote propositions.

In addition to database variables, we assume that we have another set of variable names, called *global variables*. Each one of these variables is associated with a type. The global variables are used with the assignment quantifier, in order to capture values derived from the database at different instants of time. Specifically, the global variables retain the results of queries executed on different states of the history. The use of these variables will be clear when once we define the syntax and semantics of the logic. We assume that there is a special database variable called *time*, and its value denotes the time stamp associated with the database state.

The formulas of the logic are formed using the function and relation symbols, database variables and global variables, the logical symbols $\neg, \wedge$, the assignment quantifier $\leftarrow$, square brackets $[,]$ and the temporal modal operators Until and Nexttime. In our logic, the assignment is the only quantifier. It binds a global variable to the result of a query in one of the database states of the history. A *term* is a variable or the application of a function to other terms. For example, $time + 10$ is a term; if x, y are variables and f is a binary function, then $f(x, y)$ is a term; the query BUSY-LINKS specified above is also a term. Well formed formulas of the logic are defined as follows. If $t_1, ..., t_n$ are terms of appropriate type, and R is an n-ary relational symbol, then $R(t_1, ..., t_n)$ is a well formed formula. If f and g are well formed formulas, then $\neg f$, $f \wedge g$, f Until g, Nexttime f and $([x \leftarrow t]f)$ are also well formed formulas, where x is a global variable and t is a term of the same type as x that does not contain any global variables; such a term t may represent a query on the database. A global variable x appearing in a formula is *free* if it is not in the scope of an assignment quantifier of the form $[x \leftarrow t]$. A formula without any free global variables will be called a *trigger condition*, or, in short, a *trigger*.

We present below some example formulas in the logic. If the query called $traffic(link1)$ retrieves the traffic on link1 in a communication network, then the following formula asserts that the traffic on link1 continues to be zero until it jumps to 100:

$traffic(link1) = 0$ Until $traffic(link1) = 100$.

The formula given below, which uses the assignment and the nexttime operators, indicates an update of traffic on link1:

$[x \leftarrow traffic(link1)]$ Nexttime $traffic(link1) \neq x$.

This formula should be read as follows: if the global variable x denotes the value of traffic(link1) in the present state, then the value of traffic(link1) in the next state is not equal to x. Notice how we used the global variable x to capture the value of traffic(link1) in the first state and compared it with the value of traffic(link1) in the next state. Traditional database triggers fire when a certain update occurs (e.g. the traffic on link1 increases). The above example shows that these triggers can be elegantly specified in our logic using the Nexttime and the assignment quantifier.

The syntax of PTL is similar to FTL except that PTL uses the past temporal operators Since and Lasttime .

2.3 Semantics

Intuitively, the semantics are specified in the following context. Let s_0 be the state of the database when a trigger f is activated. At each future database state s_i, the formula f is evaluated on the history starting with s_0 and ending with s_i. If f is satisfied then the trigger is *fired*, namely the user is notified or the associated action is executed; in addition, f is reactivated starting at the next database state s_{i+1}.

We define the formal semantics of our logic as follows. We assume that each type used in the logic is associated with a domain, and all the variables of that type take values from that domain. We assume a standard interpretation for all the function and relation symbols used in the logic. For example, $\leq$ denotes the standard less-than-or-equal-to relation, and $+$ denotes the standard addition on integers. Global variables capture information from various database states in the history. For this reason, we will define the satisfaction of a formula at a state on a history with respect to an evaluation, where an *evaluation* is a mapping that associates a value with each global variable. For example, consider the formula mentioned above, $[x \leftarrow traffic(link1)]$ Nexttime $traffic(link1) \neq x$, that is satisfied when the traffic on link1 is updated. The satisfaction of the subformula $traffic(link1) \neq x$ depends on the result of the query that retrieves the traffic on link1 from the current database, as well as on the value of the global variable x. The value assigned to x is the value of the traffic on link1 in the previous database state.

The definition of the semantics proceeds inductively on the structure of the formula. If the formula contains no temporal operators and no assignment (to the global variables) quantifiers, then its satisfaction at a state of the history depends exclusively on the values of the database variables in that state and on the evaluation. A formula of the form f Until g is satisfied at a state with respect to an evaluation ρ, iff one of the following two cases holds: either g is satisfied at that state, or there exists a future state in the history where g is satisfied and until then f continues to be satisfied. A formula of the form Nexttime f is satisfied at a state with respect to an evaluation, iff there is a next state and the formula f is satisfied at the next state of the history with respect to the same evaluation. Note that, for all f, the formula Nexttime f is not satisfied in the last state of a history. A formula of the form $[x \leftarrow t]f$ is satisfied at a state with respect to an evaluation, iff the formula f is satisfied at the same state with respect to a new evaluation that assigns the value of the term t to x and keeps the values of the other global variables unchanged. A formula of the form $f \wedge g$ is satisfied iff both f and g are satisfied at the same state; a formula of the form $\neg f$ is satisfied at a state iff f is not satisfied at that state.

In our formulas we use the additional propositional connectives $\vee$ (*disjunction*), $\Rightarrow$ (*logical implication*) all of which can be defined using $\neg$ and $\wedge$. We will also use the additional temporal operators Eventually and Always which are defined as follows. The temporal operator Eventually f asserts that f is satisfied at some future state, and it can be defined as *true* Until f. The temporal operator Always f asserts that f is satisfied at all future states, including the present state, and it can be defined as $\neg$ Eventually $\neg f$.

We would like to emphasize that, although the above context implies that f is evaluated at each database state, our processing algorithm avoids this overhead by specifying a set of materialized views to the database management system; the evaluation takes place only when one of the materialized views changes, i.e., only when an update that is relevant to f occurs. The algorithm performing the evaluation is given as a parameter the set of changes to the materialized views.

The semantics of the past temporal operators Since and Lasttime are defined as follows. A formula of the form f Since g is satisfied at a state with respect to an evaluation ρ, iff one

of the following two cases holds: either g is satisfied at that state, or there exists a past state in the history where g is satisfied and since then f continues to be satisfied. A formula of the form Lasttime f is satisfied at a state with respect to an evaluation, iff there is a previous state (i.e. it is not the first state) and the formula f is satisfied at the previous state of the history with respect to the same evaluation. Note that, for all f, the formula **Lasttime** f is not satisfied in the first state of a history.

2.4 Examples

The following are some examples using FTL.

The following formula, called OVERLOAD, uses the assignment quantifier and the Eventually operator. It indicates that the traffic on link1 doubles within ten minutes. The formula states that there exists a future state such that: if x, t denote the values of traffic(link1) and *time* in that state, then there exists another state after it in which the value of traffic(link1) is at least $2x$ and the time is less than or equal to $t + 10$. Here, x and t are global variables. We split the formula over two lines.

(A) Eventually $[t \leftarrow time][x \leftarrow traffic(link1)]$
(Eventually $(traffic(link1) \geq 2x \wedge time \leq t + 10)$).

Now consider a trigger that is to fire when the traffic on any link doubles, not just when the traffic on link1 doubles. Let *Traffic-on-Link* be a relation that contains link names and the traffic on each link. The new trigger can be specified in our logic by a modest modification to formula A above. Specifically, the following formula asserts that the traffic on some link doubles within ten minutes.

(B) Eventually $([t \leftarrow time]$ [$x \leftarrow$ Traffic-on-Link]
Eventually (**query1** $\wedge time \leq t + 10$)

In the above formula **query1** is a traditional database predicate. It refers to two relations x and $Traffic - on - Link$ each of which has two attributes *name* and *traffic*. This predicate evaluates to **true** if there is a tuple t in x and a tuple u in $Traffic - on - Link$ such that $u.name = t.name$ and $u.traffic \geq 2(x.traffic)$. This predicate can easily be specified in any relational query language, e.g. SQL.

In the above logic, we are explicitly using the time variable to compare the time at different points on the history. We can make the formulas more readable by omitting the time variable from the formulas, and using time constants as subscripts of temporal operators. To do so, we augment the language with bounded modal operators of the form Until$_{\leq c}$, Until$_{\geq c}$ and Until$_{=c}$, where c is some positive constant. Intuitively, f Until $_{\leq 10}$ g asserts that within ten minutes the formula g holds, and until then f continues to hold. We also use the bounded operators Eventually – within–c (g) as an abbreviation for the formula $True$ Until$_{\leq c}g$.
Formal definitions of the above operators are given below. For any formulas f and g,

f Until$_{\leq c}$ $g \equiv [t \leftarrow time](f$ Until $(g \wedge time \leq t + c))$,

The formal definitions of the other bounded operators $\mathsf{Until}_{=c}$ and $\mathsf{Until}_{\geq c}$ are similar (using $=$ and $\geq$ in place of $\leq$). Using the bounded operators, the formula (A) can be expressed as:

(C) Eventually $([x \leftarrow traffic(link1)]$
Eventually $-$ within$-$10 $(traffic(link1) = 2x))$.

Our logic is more expressive than some of the existing temporal database query languages, such as the Temporal Calculus (of Tuzhilin and Clifford). For example, consider the trigger SIMULTANEOUS-INCREASE that states the following condition: In a period of at least 50 minutes, whenever the traffic on link1 doubles in an update, then the traffic on link2 also doubles in the same update. This trigger can not be expressed in the Temporal Calculus. However, this trigger can be specified in our logic by the formula $g\,\mathsf{Until}_{\geq 50} True$ where g is the following formula.

$[x \leftarrow traffic(link1)][y \leftarrow traffic(link2)]$
Nexttime $(traffic(link1) \geq 2x \Rightarrow traffic(link2) \geq 2y)$.

The formula $g\,\mathsf{Until}_{\geq 50} True$ states that the property g is satisfied continuously for at least the next 50 minutes, where g states the following property: if x,y denote the values of traffic(link1) and traffic(link2) in the present state, then in the next state $traffic(link2) \geq 2y$ if $traffic(link1) \geq 2x$.

The above examples can also be specified in PTL. Please refer to [4] for examples of trigger condition expressed in PTL and also for the syntax and semantics of temporal aggregates. This paper also gives algorithms for processing PTL trigger conditions with aggregates. Furthermore, it discusses how composite transactions can be programmed in our rule language. It discusses how the processing algorithms can be modified to process proactive and retroactive updates.

3 Heterogeneous Databases

In a large organization such as the U.S. Airforce, different divisions/ departments may use different types of database management systems, since users within each department will choose a database management system which is most suitable for their applications. Consequently, it is rather difficult to specify a trigger that involves multiple types of databases. The problems are as follows:

(1) Users may not know different query languages which are supported by the different database management systems.

(2) The locations of various data items may not be known to the users.

(3) Query processing may be very time consuming, as data may have to be extracted from different databases before they are correlated (say using the relational "join" operation) and efficient access paths may be destroyed after the extraction process.

In a heterogeneous environment containing different types of database systems, it is difficult for users of one type of database systems to access data stored in different types of databases systems. Among the difficulties are the differences in data models and query languages. In this project, we addressed the issues of schema and query translation between object-oriented databases and relational databases. In the following subsection we sketch the schema and query translators:

3.1 Schema translation between OODB systems and Relational systems

The following steps are used in schema translation from OODB to Relational systems.

1. Transform each class to a relation. Let Rel(C) denote the relation transformed from class C. The name of Rel(C) is also C.

2. The OID of a class C becomes the primary key of the relation Rel(C) and is re-named to C-OID. All OID attributes are assigned the same data type.

3. Each non-set primitive attribute of a class C becomes an attribute of the relation Rel(C) and the domain of the class attribute becomes the data type of the corresponding relation attribute.

4. Each non-set complex attribute of a class C, which has domain class C1, is re-named to C1-OID and stays in Rel(C). Attribute C1-OID of relation Rel(C) is a foreign key from relation Rel(C1).

5. If class C has a primitive set attribute A, then exclude A from relation Rel(C) and create a new relation with name C-A. C-A has two attributes C-OID and A. The key of C-A consists of the two attributes.

6. If class C has a complex set attribute A with domain class C1, then exclude A from relation Rel(C) and create a new relation with name C-C1. C-C1 has two attributes C-OID and C1-OID. The key of C-C1 consists of the two attributes.

The following steps are used in schema translation from Relational systems to OODB systems.

1. Each relation R is transformed to a class (denoted by C(R)), augmented by an OID field. R and C(R) have the same name. Each attribute of R becomes an attribute of C(R) and the data type of a relation attribute becomes the domain class of the corresponding class attribute.

2. If a relation R1 contains a foreign key FK from relation R2, then change the current domain class of FK to C(R2). If FK contains multiple attributes, then first replace them by a single attribute. Note that since FK is a key for relation R2, there is a one-to-one correspondence between OIDs in C(R2) and attribute values of R2 under FK.

3. For two relations, say R1 and R2, if the entity type corresponding to R1 is a supertype of the entity type corresponding to R2, then let C(R1) be a superclass of C(R2) and remove from C(R2) all the attributes, with the exception of the OID attribute, which appears in C(R1).

4. For a relation R corresponding to a unary many-to-many relationship on an entity set E (which is transformed to relation R(E)), if the relationship does not have its own attribute, create two *set attributes* SA1 and SA2 for C(R(E)) and let R(E) be the domain class of the two set attributes. Discard class C(R).

5. For a relation R corresponding to a binary many-to-many relationship between two entity sets E1 and E2 (which are transformed to relations R(E1) and R(E2)), if the relationship does not have its own attribute (i.e., all attributes of R are foreign keys), create a *set attribute* for C(R(E1)) with R(E2) as its domain and, at the same time, create a set attribute for C(R(E2)) with R(E1) as its domain. Discard class C(R).

6. A ternary or higher degree relationship will be transformed to a relation which contains the keys of those relations involved.

3.2 Query Translation between OODB and Relational Systems

In the query translation process, a relational query graph is used to represent a relational query and an object-oriented query graph is used to represent an object-oriented query. A query graph consists of a set of vertices and a set of edges joining the vertices. A vertex is an instance of a relation/class referred to in the query. In case of a relational query, the edges are undirected edges and they represent join conditions. In the case of an object-oriented query, the query may have both directed and undirected edges. A directed edge (c, d) represents a traversal of the class c to the class d in a path expression in the query. An undirected edge represents a join condition since joins are also allowed in object-oriented queries.

When a relational query is received by our translator, it is first converted into a relational query graph, then the graph is converted into an object-oriented query graph and finally, the object-oriented query graph is converted into an object-oriented query. The translation of an object-oriented query goes through the reverse process i.e. it is converted into an object-oriented query graph, then a relational query graph, and finally into a relational query. We make use of a generic OODB query language having features of set and complex attributes. The details of the translations can be found in [8, 9]

4 Systems Developed

As part of this project, we built the following systems.

1. A prototype PTL trigger system on top of the Sybase Database Management System.

2. A prototype FTL trigger system on top of the Sybase Database Management System.

3. A prototype system for translating relational schemas to object oriented schemas.

4. A prototype system for translating object oriented schemas to relational schemas.

5. A prototype system for translating object oriented queries to relational queries.

6. A prototype system for translating relational queries to object oriented queries.

M. Sc. degrees completed:
Minglin Deng, "Past Temporal Logic Trigger Evaluation System in Sybase", Oct. 94.
Yi Zhang, "Translation of object-oriented queries to relational queries", Mar. 94.
Ph. D. degrees completed:
Yixiu Hwang, "Dynamic Data Replication in Distributed Systems", 1995.

References

[1] MingLin Deng, A. Prasad Sistla, O. Wolfson, *Temporal Condition Evaluation and Retroactive/Proactive Updates in ACtive Database Systems*, International Workshop on Active and Real-time Database Systems, June 1995, Skovde, Sweden.

[2] Y. Huang, R. Sloan, O. Wolfson, *Divergence Caching in Client-Server Architectures*, Proceedings of the third International Conference on Parallel and Distributed Information Systems (PDIS), Austin, TX, Sept. 1994, pp. 131-139.

[3] P. Sistla, O. Wolfson, *Temporal Triggers in Active Databases*, IEEE Transactions on Knowledge and Data Engineering (TKDE), Vol. 7(3), June 1995.

[4] P. Sistla, O. Wolfson, *Temporal Triggers and Integrity Constraints in Active Databases*, ACM SIGMOD 95.

[5] Y. Huang, A. P. Sistla, O. Wolfson, *Data Replication for Mobile Computers*, Proceedings of the ACM-SIGMOD 1994 International Conference on Management of Data, Minneapolis, Minnesotta, May 1994.

[6] Meng,W. Yu,C. and Kim,W., *A theory of translation from relational queries to hierarchical queries*, IEEE Transactions on Knowledge and Data Engineering, 1995.

[7] Meng,W. Yu,C. and Kim,W., *Processing hierarchical queries in heterogeneous environment*, IEEE Conference on Data Engineering,1992.

[8] Meng,W.Yu,C. Kim,W. Wang,G. Pham,T. and Dao,S., *Construction of relational front-end for object-oriented database systems*, IEEE Conference on Data Engineering, 1993.

[9] Yu,C., Zhang, Y., Meng, W., Kim, W., Pham, T. and Dao, S., *Translation of object-oriented queries to relational queries*, IEEE Data Engineering Conference, 1995.

[10] Yu, C., Meng, W., *Progress in Database Search Strategies*, IEEE Software 1994.

[11] Meng, W. and Yu, C. *Query Processing in Multidatabase systems*, in "Modern Database Systems", edited by W. Kim, ACM/Addison Wesley Press, 1995.